CV Programming Language: C++

Project #8 Thinning

Andres Quintero

Due Date:

Soft copy: 4/5/2020

Hard copy: 4/5/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: inFile 🡨 open from argv

outFile1, outFile2🡨 open from argv

step 1: numRows, numCols, minVal, maxVal 🡨 read from inFile

outFile1 🡨 output numRows, numCols, minVal, maxVal to outFile1

dynamically allocate firstAry of size numRows + 2 by numCols + 2.

dynamically allocate secondAry of size numRows + 2 by numCols + 2.

step 2: zeroFrame(firstAry)

zeroFrame(secondAry)

step 3: loadImage (inFile, firstAry)

step 4: prettyPrint (firstAry, outFile2) // This print is before thinning

step 5: changeFlag 🡨 0

step 6: doThinning (firstAry, secondAry, changeFlag)

Step 7: prettyPrint (firstAry, outFile2)

Step 8: repeat step 5 to step 7 while changeFlag > 0

step 9: outFile1 🡨 output firstAry from [1][1] \*without\* extra rows and cols

step 10: close all files

**bool check3n4Conditions()**

step 0: minNum <- given (either 3 or 4, depends on direction)

firstAry <- given

i, j, <- given

//checking condition 3 first

step 1: condition3Counter <- 0

step 2: checking surrounding neighbors only

if neighborVal > 0

condition3Counter++

step 3:if condition3Counter < return false

// else continue on to check for condition 4

Step 4: // check if horizontal cross will lose connectness

If (CENTER TOP && CENTER BOTTOM == 0)

If Left side and Right side BOTH have at least one object pixel

Return false

Step 5: check if vertical cross will lose connectness

If (CENTER RIGHT && CENTER LEFT == 0)

A close up of text on a black background

Description automatically generated If Top side and Bottom side BOTH have at least one object pixel

Return false

// checking corners for loss of connectness

Step 6: if TOP LEFT > 0

if( bottom && right == 0) return false;

Step 7: if TOP RIGHT > 0

if( bottom && left == 0) return false;

Step 8: if BOTTOM LEFT > 0

if( top && right == 0) return false;

Step 9: if BOTTOM RIGHT > 0

if( top && left == 0) return false;

// Passes BOTH conditions at this point, return true

step 10: return true

**Source code:**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

//DEBUGSTUFF

void printArrayDebug(int\*\* array, int rows, int cols){

for(int i = 0; i < rows; i++){

for(int j = 0; j < cols; j++){

cout << array[i][j] << " ";

}

cout << endl;

}

}

void print3x3(int\*\* Ary, int i, int j){

int count =0;

for(int a = -1; a < 2 ; a++){

for(int b = -1; b < 2; b++){

if(a == 0 && b==0){

// Dont count middle pixel

} else {

if(Ary[i+a][j+b] > 0){count++;}

}

}

}

}

// Prototypes

void zeroFrame(int\*\* array, int rows, int cols);

void loadImage(ifstream& inFile, int\*\* Ary, int rows, int cols);

void prettyPrint(int\*\* Ary, ofstream& outFile, int rows, int cols);

void doThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols);

void northThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols);

void southThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols);

void westThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols);

void eastThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols);

void copyArys(int\*\* firstAry, int\*\* secondAry, int rows, int cols);

bool check3n4Conditions(int\*\* Ary, int i, int j, int minNum);

int main(int argc, char\* argv[]){

// CLI files

ifstream inFile(argv[1]);

ofstream outFile1(argv[2]);

ofstream outFile2(argv[3]);

// Variables

// 1

int numRows, numCols, minVal, maxVal;

int changeFlag, cycleCount = 1;

inFile >> numRows;

inFile >> numCols;

inFile >> minVal;

inFile >> maxVal;

outFile1 << numRows << " " << numCols << " " << minVal << " " << maxVal << endl;

// allocating firstAry and secondAry

int \*\*firstAry = new int\*[numRows+2];

for(int i = 0; i < numRows+2; i++){

firstAry[i] = new int[numCols+2];

}

int \*\*secondAry = new int\*[numRows+2];

for(int i = 0; i < numRows+2; i++){

secondAry[i] = new int[numCols+2];

}

// 2

zeroFrame(firstAry, numRows+2, numCols+2);

zeroFrame(secondAry, numRows+2, numCols+2);

// 3

loadImage(inFile, firstAry, numRows, numCols);

// DEBUGSTUFF

// printArrayDebug(firstAry, numRows+2, numCols+2);

// print3x3(firstAry, 5, 11);

// 4

outFile2 << "prettyPrint of image before thinning: " << endl;

prettyPrint(firstAry, outFile2, numRows, numCols);

changeFlag = 1; // run atleast once

// 5 - 7

while(changeFlag > 0){

changeFlag = 0;

doThinning(firstAry, secondAry, changeFlag, numRows, numCols);

outFile2 << "Result of thinning cycle " << cycleCount-1 << ": " << endl;

prettyPrint(firstAry, outFile2, numRows, numCols);

cycleCount++;

}// 8

// 9

for(int i = 1; i < numRows+1; i++){

for(int j = 1; j < numCols+1; j++){

outFile1 << firstAry[i][j] << " ";

}

outFile1 << endl;

}

//10

inFile.close();

outFile1.close();

outFile2.close();

}

// Functions

void copyArys(int\*\* firstAry, int\*\* secondAry, int rows, int cols){

for(int i = 0; i < rows; i++){

for(int j = 0; j < cols; j++){

firstAry[i][j] = secondAry[i][j];

}

}

}

bool check3n4Conditions(int\*\* Ary, int i, int j, int minNum){

// Condition 3

int condition3Counter = 0;

for(int a = -1; a < 2 ; a++){

for(int b = -1; b < 2; b++){

if(a == 0 && b==0){

// Dont count middle pixel

} else {

if(Ary[i+a][j+b] > 0){condition3Counter++;}

}

}

}

// No need to move on to the next step if < 4 or < 3

if(condition3Counter < minNum){return false;}

// Condition 4

// Checking horizontal and vertical

if(Ary[i][j-1] == 0 && Ary[i][j+1]== 0){

int topCounter = 0, bottomCounter = 0;

for(int a = -1; a < 2; a++){

if(Ary[i-1][j+a] > 0){topCounter++;}

if(Ary[i+1][j+a] > 0){bottomCounter++;}

}

if(topCounter > 0 && bottomCounter>0){return false;}

}

if(Ary[i-1][j] == 0 && Ary[i+1][j]== 0){

int leftCounter = 0, rightCounter = 0;

for(int a = -1; a < 2; a++){

if(Ary[i+a][j-1] > 0){leftCounter++;}

if(Ary[i+a][j+1] > 0){rightCounter++;}

}

if(leftCounter > 0 && rightCounter > 0){return false;}

}

// Checking 4 corners

if(Ary[i-1][j-1] > 0){

if(Ary[i-1][j] == 0 && Ary[i][j-1] == 0) {return false;}

}

if(Ary[i-1][j+1] > 0){

if(Ary[i-1][j] == 0 && Ary[i][j+1] == 0) {return false;}

}

if(Ary[i+1][j-1] > 0){

if(Ary[i+1][j] == 0 && Ary[i][j-1] == 0) {return false;}

}

if(Ary[i+1][j+1] > 0){

if(Ary[i+1][j] == 0 && Ary[i][j+1] == 0) {return false;}

}

return true;

}

void eastThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols){

for(int i = 1; i < numRows+1; i++){

for(int j = 1; j < numCols+1; j++){

secondAry[i][j] = firstAry[i][j];

if(firstAry[i][j] > 0 && firstAry[i][j+1] <= 0){

if(check3n4Conditions(firstAry, i, j, 3) == true){

secondAry[i][j] = 0;

changeFlag++;

}

}

}

}

}

void westThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols){

for(int i = 1; i < numRows+1; i++){

for(int j = 1; j < numCols+1; j++){

secondAry[i][j] = firstAry[i][j];

if(firstAry[i][j] > 0 && firstAry[i][j-1] <= 0){

if(check3n4Conditions(firstAry, i, j, 3) == true){

secondAry[i][j] = 0;

changeFlag++;

}

}

}

}

}

void southThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols){

for(int i = 1; i < numRows+1; i++){

for(int j = 1; j < numCols+1; j++){

secondAry[i][j] = firstAry[i][j];

if(firstAry[i][j] > 0 && firstAry[i+1][j] <= 0){

if(check3n4Conditions(firstAry, i, j, 4) == true){

secondAry[i][j] = 0;

changeFlag++;

}

}

}

}

}

void northThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols){

for(int i = 1; i < numRows+1; i++){

for(int j = 1; j < numCols+1; j++){

secondAry[i][j] = firstAry[i][j];

if(firstAry[i][j] > 0 && firstAry[i-1][j] <= 0){

if(check3n4Conditions(firstAry, i, j, 4) == true){

secondAry[i][j] = 0;

changeFlag++;

}

}

}

}

}

void doThinning(int\*\* firstAry, int\*\* secondAry, int& changeFlag, int numRows, int numCols){

// 1

northThinning(firstAry, secondAry, changeFlag, numRows, numCols);

copyArys(firstAry, secondAry, numRows+2, numCols+2);

// 2

southThinning(firstAry, secondAry, changeFlag, numRows, numCols);

copyArys(firstAry, secondAry, numRows+2, numCols+2);

// 3

westThinning(firstAry, secondAry, changeFlag, numRows, numCols);

copyArys(firstAry, secondAry, numRows+2, numCols+2);

// 4

eastThinning(firstAry, secondAry, changeFlag, numRows, numCols);

copyArys(firstAry, secondAry, numRows+2, numCols+2);

}

void prettyPrint(int\*\* Ary, ofstream& outFile, int rows, int cols){

for(int i = 1; i < rows+1; i++){

for(int j = 1; j < cols+1; j++){

if(Ary[i][j] > 0){

outFile << Ary[i][j] << " ";

} else {

outFile << " ";

}

}

outFile << endl;

}

}

void loadImage(ifstream& inFile, int\*\* Ary, int rows, int cols){

int value;

for(int i = 1; i < rows+1; i++){

for(int j = 1; j < cols+1; j++){

inFile >> value;

Ary[i][j] = value;

}

}

}

void zeroFrame(int\*\* array, int rows, int cols){

for(int i = 0; i < rows; i++){

for(int j = 0; j < cols; j++){

array[i][j] = 0;

}

}

}

**Image1 outputs**

**outFile1**

**A close up of a logo

Description automatically generated**

**outFile2**

**A close up of a piece of paper

Description automatically generated**

**outFile2 cont.**

**A screenshot of a cell phone

Description automatically generated**

**END OF image1 OUTPUTS**

**Image2 outputs**

**outFile1**

**A close up of a logo

Description automatically generated**

**outFile2**

**A screenshot of a computer

Description automatically generated**

**outFile2 cont.**

**A screenshot of a cell phone

Description automatically generated**

**outFile2 cont.**

**A screenshot of a cell phone

Description automatically generated**

**END OF image2 OUTPUTS**